# USB Compliance Checklist

**Revision 1.0**

**June 6, 1996**

# Introduction

This is a checklist to help in design reviews of ASICs, Components, and Peripherals to check their compliance with Universal Serial Bus Specification, Revision 1.0.

This checklist is also used to qualify a USB product for the System Integrator List by creating a paper trail of testing for USB compliance.

# *USB Device Product Information*

| | |
|---|---|
| Date | |
| Vendor Name | |
| Vendor Street Address | |
| Vendor City, State, Zip | |
| Vendor Phone Number | |
| Vendor Contact, Title | |
| Product Name | |
| Product Model Number | |
| Product Revision Level | |

# USB Protocol Checklist

The reference setup for this checklist is a USB link with one USB agent on each side of it. This checklist specifies behavior expected from these USB agents to satisfy chapter 8 of the USB protocol. One of these two agents is the controlling agent (host or hub) and the other is the controlled agent (device or hub). Except for a few noted items, both types of agents need to adhere to the same requirements. The hub is a special entity because its upstream ports act like a device and its downstream ports act like a host. So for the purposes of this checklist, the terms host and hub downstream port are equivalent as are the terms device and hub upstream/root port. Hub specific requirements are listed in the hub chapter.

This checklist is organized in two broad sections. The first section lists expected or normal behavior from an USB agent when it drives the bus or receives from the bus . So tests in this section need to be done in two parts - ensure that the agent drives the correct signals and ensure that the agent accepts correct signaling (correctness by design and correctness by test). The second section lists expected behavior of an USB agent when it sees an error or abnormal condition on the bus (robustness by test concept). Any deviation from this checklist should be noted and explained in the explanations area. These lists are organized starting with the smallest signal entity on the bus and ending at the transfer level.

Some definitions of terms used in the checklist. Note that some terms are overloaded (e.g. data) and definition should be derived from context. Bit order is from left to right unless specified otherwise.

**Sense**: The relationship between D+ and D- voltages for J state on a USB link e.g. on a fullspeed link, D+ > D- ; on a low-speed link D+ < D-.

**NZB**: NrZ bit; smallest data structure referenced in spec; represented as 1, 0 and X (single-ended 0) e.g. 1

**Field**: the next higher form of data structure; can be represented as NIBs or NZBs e.g. sync field is KJKJKJKK or 00000001

**Packet**: is the most referenced data structure; is built out of fields e.g. Token packet

**Phase**: of a transaction is 0 or 1 packet e.g. token phase

**Transaction**: is the basic unit for unidirectional data transfer and is a set of packets; for unidirectional endpoints (bulk, interrupt, iso) this is the final level of communication with protocol implications e.g. OUT transaction

**Stage**: is a set of one or more unidirectional transactions e.g. data stage

**Transfer**: is a unit of bi-directional data transfer and is built out of stages. It is used by bi-directional endpoints only. e.g. control transfer

**Turnaround time**: time between an agent seeing the EOP and starting to drive the bus

**Timeout period**: amount of time that an agent awaiting a response waits before invalidating the transaction

**Target**: could be a pipe in the host or endpoint in a device.

The addendum lists test scenarios which can be used to develop tests for the items in the checklist as well as some useful Perl programs.

# Design-and-Test section:

## Bitstream

A bitstream is a set of bits from J,K, 0 (nrzi) or 0,1,S(nrz)

| Name | Test description | Spec sec | Status |
|---|---|---|---|
| BSD1 | Is >=2.5 us of NIB 0 anywhere in a bitstream recognized on the root port of a device as a USB RESET? | 7.1.4.3 | yes ___ no___ |
| BSD2 | Is >=2.5 us of NIB 0 anywhere in a bitstream recognized by the host or the downstream port of a hub as a disconnect event? | 7.1.4.1 | yes ___ no___ |
| BSD3 | Is >=2.5 us of NIB J recognized by the host or a downstream port of a hub as a connect event? | 7.1.4.1 | yes ___ no___ |
| BSD4 | Is >=1 ms of NIB K followed by an EOP recognized as an end of resume event by a target? | 11.5.1 | yes ___ no___ |
| BSD5 | Does any transition from the J state on the root port of a suspended device wakeup the device? | 7.1.4.5 | yes ___ no___ |
| BSD6 | Is the possibility of a NIB 1 <1 bit time during bus transitions accounted for? | 7.1.13 | yes ___ no___ |
| BSD7 | Is the sense of signaling on a link either full-speed or low-speed but not both? | 11.2.5 | yes ___ no___ |
| BSD8 | Does the sense of signaling on a link correspond to speed of link? | 7.1.4 | yes ___ no___ |
| BSD9 | Is the bitstream on the bus nrzi encoded? | 7.1.5 | yes ___ no___ |
| BSD10 | Does the bitstream on the bus implement bit stuffing prior to transmission? | 7.1.6 | yes ___ no___ |
| BSD11 | Does the CRC bitstream on the bus implement bit stuffing? | 8.3.5 | yes ___ no___ |
| BSD12 | Is bit stuffing implemented even if a stuffed bit is required after the last bit of the packet? | 8.3.5 | yes ___ no___ |
| BSD13 | Is bit stuffing done after the CRC computation on the transmitted bit stream? | 8.3.5 | yes ___ no___ |
| BSD14 | Is nrzi encoding done after the bit stuffing on the transmitted bit stream? | 7.1.6 | yes ___ no___ |
| BSD15 | Is nrzi->nrz decoding done before bit unstuffing? | 7.1.6 | yes ___ no___ |
| BSD16 | Is bit unstuffing done before the bitstream is parsed? | 7.1.6 | yes ___ no___ |

## Field

A field can be

sync - 8 bit field with NZB value 00000001
PID - listed in Table 8-1 of spec
address - 7 bit field
endpoint - 4 bit field
frame number - 11 bit field
token CRC - 5 bit field
data - 0 to 1023 byte field
data CRC - 16 bit field
EOP - 3 bit field with NIB value 00J

| Name | Test description | Spec sec | Status |
|---|---|---|---|
| FLD1 | Is the sync field as measured on the bus wires correct i.e NIB KJKJKJKK? | 8.2 | yes ___ no___ |
| FLD2 | Is the packet type in the PID one of those listed in Table 8-1 | 8.3.1 | yes ___ no___ |
| FLD3 | Is the PID check the one's complement of the packet type field | 8.3.1 | yes ___ no___ |
| FLD4 | Is the token CRC generated with the polynomial NZB 00101 | 8.3.5.1 | yes ___ no___ |
| FLD5 | Does the CRC computation on a token/SOF bitstream leave a residual of NZB 01100 at the EOP | 8.3.5.1 | yes ___ no___ |

| | | | |
|---|---|---|---|
| FLD6 | Is the data CRC generated with the polynomial NZB1000000000000101 | 8.3.5.2 | yes ___ no___ |
| FLD7 | Does the CRC computation on a data packet bitstream leave a residual of NZB 1000000000001101 at the EOP | 8.3.5.2 | yes ___ no___ |
| FLD8 | Is the 7-bit address field sent out LSB first on the bus | 8.3.2.1 | yes ___ no___ |
| FLD9 | Is the 4-bit endpoint field sent out LSB first on the bus | 8.3.2.2 | yes ___ no___ |
| FLD10 | Is the 11-bit frame number field sent out LSB first on the bus | 8.3.3 | yes ___ no___ |
| FLD11 | Is each data byte in the data field sent out LSB first | 8.3.4 | yes ___ no___ |
| FLD12 | Is the CRC shift register contents inverted to form the CRC field | 8.3.5 | yes ___ no___ |
| FLD13 | Is the CRC field sent out MSB first | 8.3.5 | yes ___ no___ |
| FLD14 | Is the EOP correctly consituted i.e NIB 00J | 7.1.11.2 | yes ___ no___ |
| FLD15 | Does a full-speed receiver recognize 82ns to 2.5 us of NIB 0 followed by a J transition as a valid EOP | 7.1.12 | yes ___ no___ |
| FLD16 | Does a low-speed receiver recognize 670 ns to 2.5 us of NIB 0 followed by a J transition as a valid EOP | 7.1.12 | yes ___ no___ |
| FLD17 | Does a low speed device recognize low-speed keepalive strobes | 11.2.5.1 | yes ___ no___ |

## Packet

A packet is made up of fields which are formatted as described in sec. 8.4 of spec and can be one of the following:

PRE - sync pid
SOF - sync pid timestamp tokenCRC eop
Token - sync pid addr endpt tokenCRC eop
data - sync pid data..... data dataCRC eop
handshake - sync pid eop

| **Name** | **Test description** | Spec sec | Status |
|---|---|---|---|
| PKD1 | Is the PRE packet 16 bits long | 8.6.5 | yes ___ no___ |
| PKD2 | Is the PRE packet constituted as sync followed by PID | 8.6.5 | yes ___ no___ |
| PKD3 | Is the Token packet 32 bits + EOP | 8.4.1 | yes ___ no___ |
| PKD4 | Is the token constituted as sync followed by PID followed by address followed by endpoint followed by token CRC followed by EOP | 8.4.1 | yes ___ no___ |
| PKD5 | Is the SOF packet 32 bits + EOP | 8.4.2 | yes ___ no___ |
| PKD6 | Is the SOF constituted as sync followed by PID followed by frame number followed by token CRC followed by EOP | 8.4.2 | yes ___ no___ |
| PKD7 | Is the handshake packet 16 bits + EOP | 8.4.4 | yes ___ no___ |
| PKD8 | Is the handshake constituted as sync followed by PID followed by EOP | 8.4.4 | yes ___ no___ |
| PKD9 | Is the data packet an integral number of bytes ( 4 to 1027) + EOP | 8.4.3 | yes ___ no___ |
| PKD10 | Is the data packet constituted as sync followed by PID followed by 0 to 1023 bytes of data followed by data CRC followed by EOP | 8.4.3 | yes ___ no___ |
| PKD11 | Is the data payload of a low speed packet limited to 8 bytes | 8.6.5 | yes ___ no___ |

## Transaction

Transactions are sets of packets used for unidirectional data transfer and can be one of the following: (host phase in italics; device phase in regular)
*SOF*
*setup data* ack
*out data* ack/nak/stall
*out data*
*in* data *ack*
*in* data/nak/stall
*pre setup pre data* ack
*pre out pre data* ack/nak/stall
*pre in* data *pre ack*

| Name | Test description | Spec sec | Status |
|---|---|---|---|
| TRD1 | Does the device implement default address of 0 on device reset | 8.3.2.1 | yes ___ no___ |
| TRD2 | Does the device implement a bidirectional control endpoint 0 for every address | 8.3.2.2 | yes ___ no___ |
| TRD3 | Does the generated packet fit the phase of the transaction as listed above | 8.5,8.6.5 | yes ___ no___ |
| TRD4 | Is the turnaround time of a packet-sourcing agent greater than 2 bit times | 7.1.15 | yes ___ no___ |
| TRD5 | Is the turnaround time of a packet-sourcing agent less than 6.5 (7.5 with integrated cable) bit times | 7.1.15 | yes ___ no___ |
| TRD6 | Is the timeout period at an agent awaiting response greater than 16 bit times | 7.1.16 | yes ___ no___ |
| TRD7 | Is the timeout period at an agent awaiting response less than 18 bit times | 7.1.16 | yes ___ no___ |
| TRD8 | Is an unsuccessful (NAK or timeout in non-token phase) transaction retried | 8.6.2-4 | yes ___ no___ |
| TRD9 | Does the retried transaction use the same data PID as the original transaction | 8.6.2-4 | yes ___ no___ |
| TRD10 | Do interrupt endpoints used in rate feedback mode toggle the sequence bit without regard to presence or type of handshake | 8.5.3 | yes ___ no___ |
| TRD11 | Do handshakes conform to order of precedence detailed in tables of sec. 8.4.5 | 8.4.5 | yes ___ no___ |
| TRD12 | Are low speed transactions limited to those needed to support interrupt and control endpoints | 8.6.5 | yes ___ no___ |

## Transfer

Transfers are data structures used by bidirectional (control endpoints). The transfer is made up of stages which are sets of unidirectional transactions. They can be one of:
setup0 *out1 out0 out1 ... out0/1* in1
setup0 *in1 in0 in1 ... in0/1* out1
setup0 in1
Transactions in italics constitute the data stage ; there may or may not be a data stage between the setup stage and status stage. Suffix of 0 or 1 indicates the data PID used in the transaction

| Name | Test description | Spec sec | Status |
|---|---|---|---|
| TFD1 | Does the setup stage use a data0 PID | 8.5.2 | yes ___ no___ |
| TFD2 | Does the status stage use a data1 PID | 8.5.2 | yes ___ no___ |
| TFD3 | Does the data stage always start with a data1 PID | 8.5.2 | yes ___ no___ |
| TFD4 | Are all the transactions of the data stage in the same direction | 8.5.2 | yes ___ no___ |
| TFD5 | Is there a change of direction when entering the status change | 8.5.2 | yes ___ no___ |
| TFD6 | Is the data packet used in the status stage 0 bytes in length | 8.5.2 | yes ___ no___ |

# Test for robustness Section:

## Bitstream

A compliant bitstream is a set of bits from J,K,0 (nrzi) or 0,1,X(nrz)

| Name | Test description | N/A | Status |
|---|---|---|---|
| BST1 | Is a single ended NIB 1 of >= 1 bit time ignored by the target | | yes ___ no___ |
| BST2 | Does an agent ignore a truncated (upto 50%) first bit of the sync field without impacting the rest of the bitstream | | yes ___ no___ |
| BST3 | Is the state of the differential receiver ignored during single ended signal state | | yes ___ no___ |
| BST4 | Does the target reject bitstreams of length <1 bit time without impacting future transactions | | yes ___ no___ |
| BST5 | does the target adjust to the difference in frequency and phase between incoming clockand its internal clock | | yes ___ no___ |
| BST6 | Is a packet with a bit-stuff error rejected by the target | | yes ___ no___ |
| BST7 | Is a bitstream (which is not part of a packet) with bit stuff error ignored by the target | | yes ___ no___ |
| BST8 | Does the target reject packets with bit stuff error at the last bit of the packet | | yes ___ no___ |
| BST9 | Is bit stuffing implemented even if stuffed bit is after the last bit of the packet | | yes ___ no___ |

## Field

A compliant field can be one of:

sync - 8 bit field with NZB value 00000001
PID - listed in Table 8-1 of spec
address - 7 bit field
endpoint - 4 bit field
frame number - 11 bit field
token CRC - 5 bit field
data - 0 to 1023 byte field
data CRC - 16 bit field
EOP - 3 bit field with NIB value 00J

| Name | Test description | N/A | Status |
|---|---|---|---|
| FLT1 | Is the sync field recognized as valid even if up to two initial bits of it are corrupted (Actually, only the last 3 bits (JKK) need to be decoded). | | yes ___ no___ |
| FLT2 | Is a packet with packet type not listed in Table 8-1 ignored by the target | | yes ___ no___ |
| FLT3 | Is a packet with corrupt PID (PID check error) ignored by the target | | yes ___ no___ |
| FLT4 | Is a token with bad CRC ignored by the target | | yes ___ no___ |
| FLT5 | Is a CRC error on a data packet recognized by the target | | yes ___ no___ |
| FLT6 | Does a full speed receiver reject a NIB 0 of duration less than 40 ns as part of an EOP | | yes ___ no___ |
| FLT7 | Does a low speed receiver reject a NIB 0 of duration less than 330 ns as part of an EOP | | yes ___ no___ |

## Packet

A compliant packet is made up of fields which are formatted as described in sec. 8.4 of spec and can be one of the following:

PRE - sync pid
SOF - sync pid timestamp tokenCRC eop
Token - sync pid addr endpt tokenCRC eop
data - sync pid data..... data dataCRC eop
handshake - sync pid eop

| Name | Test description | N/A | Status |
|---|---|---|---|
| PKT1 | Is a token whose address field doesn't match any address in the device ignored by the device | | yes ___ no___ |
| PKT2 | Is a token whose endpoint field doesn't match any endpoint in the address ignored by the device | | yes ___ no___ |
| PKT3 | Is a token which doesn't match direction of endpoint ignored by the device | | yes ___ no___ |
| PKT4 | Is a SETUP token to a unidirectional endpoint ignored by the device | | yes ___ no___ |
| PKT5 | Is every endpoint capable of handling 0 length data packet in its assigned direction(s) | | yes ___ no___ |
| PKT6 | Does an ISO endpoint use 0 length data packet if fresh frame data is not available | | yes ___ no___ |
| PKT7 | Is a packet whose length doesn't match standard length for packet type rejected by target | | yes ___ no___ |
| PKT8 | Does the measurement of packet length take into account the possibility of jitter in the EOP | | yes ___ no___ |
| PKT9 | Is a bitstream not constituted according to packet rules described in last section rejected by the target | | yes ___ no___ |

## Transaction

A compliant transaction can be one of the following: (host phase in italics; device phase in regular)
*SOF*
*setup data* ack
*out data* ack/nak/stall
*out data*
*in* data *ack*
*in* data/nak/stall
*pre setup pre data* ack
*pre out pre data* ack/nak/stall
*pre in* data *pre ack*

| Name | Test description | N/A | Status |
|---|---|---|---|
| TRT1 | Is a packet which doesn't fit the current phase of a transaction rejected by the target | | yes ___ no___ |
| TRT2 | Does the receipt of a token always start a new transaction (and end a pending transaction) | | yes ___ no___ |
| TRT3 | Does a target ignore data packet with same PID as previous data packet to the endpoint but successfully complete (ack) the transaction | | yes ___ no___ |
| TRT4 | Does a timeout or error in any phase cause the transaction to be terminated | | yes ___ no___ |
| TRT5 | Is a transaction always started with a token | | yes ___ no___ |
| TRT6 | Is the data toggle implemented independently for each unidirectional endpoint | | yes ___ no___ |
| TRT7 | Does the source of ISO data ignore handshake without impacting subsequent transactions | | yes ___ no___ |

| | | |
|---|---|---|
| TRT8 | Can the target handle consecutive packets in the same direction with >=2 bit times of interpacket gap | yes ___ no___ |

## Transfer

Transfers are data structures used by bidirectional (control endpoints). The transfer is made up of stages which are sets of unidirectional transactions. A compliant transfer can be one of:

setup0 *out1 out0 out1 ... out0/1* in1
setup0 *in1 in0 in1 ... in0/1* out1
setup0 in1

Transactions in italics constitute the data stage ; there may or may not be a data stage between the setup stage and status stage. Suffix of 0 or 1 indicates the data PID used in the transaction

| Name | Test description | N/A | Status |
|---|---|---|---|
| TFT1 | Does a target receiving an unexpected data PID ignore the data but still successfully complete(ack) the transaction | | yes ___ no___ |
| TFT2 | Does the receipt of a non-zero length data packet in the status stage cause the transfer to be terminated with an error indication | | yes ___ no___ |

# USB Signals and Timing Checklist

## General Rules:

| | | |
|---|---|---|
| EL1 | Single-ended receivers recognize voltage below 0.8 V as a logic low? | yes__ no__ |
| EL2 | Single-ended receivers recognize voltage above 2.0 V as a logic high? | yes__ no__ |
| EL3 | Differential receivers recognize differential voltages of 200 mV between 0.8 and 2.5 volts? | yes__ no__ |
| EL4 | Do active data line outputs drive to 2.8 volts with a 15 KΩ load to ground? | yes__ no__ |
| EL5 | Do active data line outputs drive to 0.3 volts with a 1.5 KΩ load to 3.6 volts? | yes__ no__ |
| EL6 | Capacitance on each data line is a maximum of 20 pF (without cable)? | yes__ no__ |
| EL7 | Does the device accept a truncated bit time (down to half of a bit time) as the first bit of the SYNC field? | yes__ no__ |
| EL8 | Does the device recognize a single-ended zero of 2.5 µs or greater on its root port as a device reset? | yes__ no__ |
| EL9 | Does the device recognize any non-idle state on its root port as a resume signal? | yes__ no__ |
| EL10 | Do all downstream ports sink 200 µA ±5% at 3.0V when not driving the bus (15 KΩ resistor to ground present)? | yes__ no__ n/a__ |
| EL11 | Are open downstream ports pulled to ground? | yes__ no__ n/a__ |
| EL12 | Does the device recognize a single-ended zero of 2.5 µs or greater on any of its downstream ports as a disconnect? | yes__ no__ n/a__ |
| EL13 | Does the device recognize a non single-ended zero of 2.5 µs or greater on any of its downstream ports as a connect? | yes__ no__ n/a__ |

## Driver Port Characteristics - Full Speed Ports: This section is N/A ____.

Applicable to any port which can operate at 12 Mb/s, up or downstream.
This includes the host, full speed devices and all hub ports, including the root port.

| | | |
|---|---|---|
| EL14 | Is the pull-up source resistance between 28 and 43 Ω? | yes__ no__ |
| EL15 | Is the pull-down source resistance between 28 and 43 Ω? | yes__ no__ |
| EL16 | Data line rise times are greater than 4.0 ns and less than 20 ns? | yes__ no__ |
| EL17 | Data line fall times are greater than 4.0 ns and less than 20 ns? | yes__ no__ |
| EL18 | Are the rise and fall times matched to within 10%? | yes__ no__ |
| EL19 | Do the data lines cross over each other between 1.3 and 2.0 volts? | yes__ no__ |
| EL20 | Is the bus idle state D+ between 3.0 and 3.6V and D- at ground? | yes__ no__ |
| EL21 | Is there a 1.5 KΩ pull-up resistor on the D+ data line of the device's upstream (root) port? With Vbus present on the device and driver off: | |
| | - Does D+ source a maximum of 2.6 mA when it is grounded? | yes__ no__ |
| | - Does the pull-up resistor D+ pull up to a voltage between 3.0 and 3.6 volts with no load? | yes__ no__ |
| EL22 | When Vbus is disconnected from the device, does the D+ source no current? | yes__ no__ |
| EL23 | When Vbus is present on the device, is the magnitude of the leakage current (device not driving) from the D- data line less than 10 μA for input voltages between 0.0 and 3.3 volts? | yes__ no__ |

## Driver Port Characteristics - Low Speed Ports: This section is N/A ____.

Applicable to any port which can operate at 1.5 Mb/s, up or downstream.
This includes the host, low speed devices and all downstream hub ports.

| | | |
|---|---|---|
| EL24 | Data line rise times are greater than 75 ns and less than 300 ns? | yes__ no__ |
| EL25 | Data line fall times are greater than 75 ns and less than 300 ns? | yes__ no__ |
| EL26 | Are the rise and fall times matched to within 20%? | yes__ no__ |
| EL27 | Do the data lines cross over each other between 1.3 and 2.0 volts? | yes__ no__ |
| EL28 | Is the bus idle state D- between 3.0 and 3.6V and D+ at ground? | yes__ no__ |
| EL29 | Is there a 1.5 KΩ pull-up resistor on the D- data line of the device's upstream (root) port? Does D- source a maximum of 2.6 mA when it is grounded? | yes__ no__ |
| EL30 | When Vbus is present on the device, does D- pull-up to a voltage between 3.0 and 3.6 volts with no load? | yes__ no__ |
| EL31 | When Vbus is disconnected from the device, does the D- source no current? | yes__ no__ |
| EL32 | When Vbus is present on the device, is the magnitude of the leakage current (device not driving) from the D+ data line less than 10 μA for input voltages between 0.0 and 3.3 volts? | yes__ no__ |

## Data Source Timings - Full Speed Ports: This section is N/A____.

Applicable to a device operating at 12 Mb/s, up or downstream which acts as the source of data. This includes the host, full speed devices and the root hub port when the hub or embedded function is the addressed device.

| | | |
|---|---|---|
| EL38 | Is the transmission data rate between 11.97 and 12.03 Mb/s? | yes__ no__ |
| EL39 | Is the differential driver jitter less than ±3.5 ns? | yes__ no__ |
| EL40 | Is the differential driver jitter for paired transitions[3] less than ±4.0 ns? | yes__ no__ |
| EL41 | Is the EOP width between 160 ns and 175 ns? | yes__ no__ |
| EL42 | Is the timing skew due to the transition to the EOP on the last differential bit(s) between -2.0 ns and 5.0 ns? | yes__ no__ |
| EL43 | Is the receiver data jitter tolerance at least ± 18.5 ns? | yes__ no__ |
| EL44 | Is the receiver jitter tolerance for paired transitions[3] at least ± 9.0 ns? | yes__ no__ |
| EL45 | Does the device accept a single-ended zero of 82 ns as an EOP? | yes__ no__ |
| EL46 | Does the device reject as an EOP a single-ended zero of 40 ns or less? | yes__ no__ |

## Data Source Timings - Low Speed Ports: This section is N/A____.

Applicable to a device operating at 1.5 Mb/s, up or downstream. This includes the host and low speed devices.

| | | |
|---|---|---|
| EL47 | Is the transmission data rate between 1.4775 and 1.5225 Mb/s? | yes__ no__ |
| EL48 | Is the differential driver jitter less than ±95 ns? (n/a for host devices) | yes__ no__ n/a__ |
| EL49 | Is the differential driver jitter for paired transitions[3] less than ±150 ns? (n/a for host devices) | yes__ no__ n/a__ |
| EL50 | Is the EOP width between 1.25 μs and 1.5 μs? | yes__ no__ |
| EL51 | Is the timing skew due to the transition to the EOP on the last differential bit(s) between -40 ns and 100 ns? | yes__ no__ |
| EL52 | Is the receiver data jitter tolerance at least ± 75 ns? (n/a for host devices) | yes__ no__ n/a__ |
| EL53 | Is the receiver jitter tolerance for paired transitions[3] at least ± 45 ns? ( n/a for host devices) | yes__ no__ n/a__ |
| EL54 | Does the device accept a single-ended zero of 675 ns as an EOP? | yes__ no__ |
| EL55 | Does the device reject as an EOP a single-ended zero of 330 ns or less? | yes__ no__ |

## Full Speed Cable Characteristics: This section is N/A____.

Applicable to any cable attached (or attachable) to the upstream (root) port of a full speed device or hub.

| | | |
|---|---|---|
| EL70 | Is the cable shielded per recommendation[4] or better? | yes__ no__ |
| EL71 | Is the cable impedance between 76.5 Ω and 103.5 Ω? | yes__ no__ |
| EL72 | Is the cable delay 30 ns or less? | yes__ no__ |

## Notes:

1. All of the items in this checklist have been taken from Chapter 7 of the USB Specification, Rev. 1.0, in particular, Section 7.3. These specifications are explained in Section 7.1.
2. All voltages are referenced from the USB ground of the device being tested.
3. See USB Specification Rev. 1.0, Chapter 7, Section 7.1.11.1 for explanation of paired transitions.
4. Shielding recommended: Aluminized mylar with a 28 AWG drain wire and 65% minimum coverage, tinned copper mesh over the foil.

## Explanations:

This section should be used to explain any "no" or "n/a" answers or clarify any answers on checklist items above.
Please key explanation to item number.

# USB Power

## *USB Peripheral Power Checklist*

This checklist applies to the following USB Device Manufacturer: _____

## A: General rules

| | | |
|---|---|---|
| PWR1 | Does the USB peripheral drive no power upstream over the power cable pair?. | yes___ no___ |
| PWR2 | Does the USB peripheral enumerate with bus voltages over the range of 4.40V to 5.25V? | yes___ no___ |
| PWR3 | Does the peripheral have its ground connected to the ground line of the USB cable? | yes___ no___ |
| PWR4 | Does the USB peripheral have its connector shell tied to the PC board ground plane? | yes___ no___ |
| PWR5 | Does the peripheral have one and only one root port? | yes___ no___ |

## B: Self Powered Devices

### 1: Root port characteristics (except Host)

| | | |
|---|---|---|
| PWR6 | All self powered devices must connect their ground to that of the USB cable. | yes___ no___ |
| PWR7 | Self powered devices may optionally connect their Vbus to that of the USB cable. | yes___ no___ |

### Inrush current

| | | |
|---|---|---|
| PWR8 | Does the root port present a maximum capacitance to the bus of 10 µf ? | yes___ no___ na___ |
| PWR9 | If the actual capacitance is > 10 mf, is there some type of current limiting make the inrush current characteristics of the device equivalent to that of a max capacitive load of 10 µf? | yes___ no___ |

### Max input current

| | | |
|---|---|---|
| PWR10 | Does a low power device draw less than 100 ma current at all times? | yes___ no___ na___ |
| PWR11 | Does a high power device draw less than 100 ma before and during enumeration? | yes___ no___ na___ |
| PWR12 | Does a high power device draw less than 500 ma at all times? | yes___ no___ na___ |

### $V_{BUS}$ Voltage limits

| | | |
|---|---|---|
| PWR13 | If it draws power from the bus, does a low power device operate with Vbus between 4.40 and 5.25V? | yes___ no___ na___ |
| PWR14 | If it draws power from the bus, does a full powered USB devices enumerate with Vbus between 4.40 and 5.25V? | yes___ no___ na___ |
| PWR15 | If it draws power from the bus and draws more than 100 ma during operation, does a full powered USB device operate with Vbus between 4.75 and 5.25V. | yes___ no___ na___ |

### Safety Requirements

| | | |
|---|---|---|
| PWR22 | If the device has a 2 prong line connector, is the power supply double insulated? | yes___ no___ na___ |
| PWR23 | If the device has a 3 prong line connector does the line ground connect to the device's chassis | yes___ no___ na___ |
| PWR24 | If the device has a 3 prong line connector can the ground wire sustain a 30 Amp short for 2 minutes? | yes___ no___ na___ |
| PWR25 | If the device has a metal chassis, is the chassis connected to digital ground? | yes___ no___ na___ |

# C: Bus Powered Devices

| | | |
|---|---|---|
| PWR30 | Does Vbus of the USB cable connect to the power plane of the peripheral?. | yes___ no___ |
| PWR31 | Does ground of the USB cable connect to the ground plane of the peripheral?. | yes___ no___ |

### 1: Root port

### Inrush current

| | | |
|---|---|---|
| PWR32 | Is the inrush current equivalent to a max of 10 uf capacitor across Vbus and gnd? | yes___ no___ |
| PWR33 | If the actual capacitance across Vbus and gnd is > 10 mf, is there some type of current limiter that makes the electrical characteristics of the device equivalent to a 10 µf capacitor? | yes___ no___ |

### 2: Low Power Devices

| | | |
|---|---|---|
| PWR34 | Does a low power device draw less than 100 ma at all times from the upstream bus? | yes___ no___ |
| PWR35 | Does the low power device have no downstream USB ports? | yes___ no___ |

### Max input current

| | | |
|---|---|---|
| PWR36 | Does a low power device draw less than 100 ma at any time from Vbus and gnd? | yes___ no___ |

### $V_{BUS}$ Voltage limits

| | | |
|---|---|---|
| PWR37 | Does a low powered device enumerate with Vbus between 4.40 and 5.25V measured at upstream port. | yes___ no___ |
| PWR38 | Does a low powered device operate with Vbus between 4.40V and 5.25V measured at upstream port. | yes___ no___ |

### 3: High power device

### Max input current

| | | |
|---|---|---|
| PWR39 | Is the maximum current drawn by high power device always less than 500 ma | yes___ no___ |
| PWR40 | Does a high power device enumerate drawing a max of 100 ma | yes___ no___ |

### $V_{BUS}$ Voltage limits

| | | |
|---|---|---|
| PWR41 | Does a high power devices enumerate with Vbus between 4.40V and 5.25V measured at upstream port | yes___ no___ |
| PWR42 | Does a high power device operate with Vbus between 4.75 and 5.25V as measured at upstream port | yes___ no___ |

# D. Suspend / Resume for all devices

| | | |
|---|---|---|
| PWR56 | Does a USB device enter suspend when 3.0ms or more of continuous J state exists on the bus? | yes ___ no___ |
| PWR57 | When in the suspend state does a device draw no more than 500 µa, including current drawn by termination resistors? | yes ___ no___ |
| PWR59 | Can the power supply of a self powered device maintain Vcc within spec when the device is resumed? | yes___ no___ na___ |

## Explanations:

This section should be used to clarify any answers on checklist items above. Please key explanation to item number.

ADDENDUM to USB 1.0 Protocol Compliance Checklist

Introduction
The USB 1.0 Protocol Compliance Checklist specifies a series of tests which can be used to determine if an USB agent complies with the spec. The tests in the checklist are targeted for a simulation environment but they could be used for hardware test as well. In a simulation environment, one of the two USB agents on the reference USB link would be the agent under test and the other would typically be a bus functional model . It would also be beneficial to have a monitor which observes the signaling on the bus and reports the activity and errors if any.

As described in the checklist, each item in the first section should have at least two tests - the first should test if the agent generates the correct signaling; the second should test if the agent accepts correct signaling e.g. can the agent generate a correct sync pattern and can the agent recognize a signal pattern on the bus as sync if the pattern matches requirements in the spec. As one moves through the checklist, the data structures become more complex and the number of tests per item will also increase e.g. each test in the packet sub-section should be implemented for each packet type. In items where a range is specified, some means to parametrically vary the test through the range should be used e.g. shmoo testing.

The checklist has tried to identify all the corner cases in the spec. Each design will introduce its own design corners which will need to be tested as well to ensure a robust design.

Test scenarios:
BSD1-2 : include tests where the NIB 0 stream starts anywhere within any packet or in any inter-packet region of any transaction or transfer
BSD5 : include transitions from NIB J to NIB 0, NIB K
BSD11-16 : include cases of bit stuffing after last bit of crc; and cases of bit stuffing just before the last bit of crc which can be either 0 or 1; include similar tests for unstuffing as well.
FLD15-16 : include cases which test that the EOP 'jitter' (fractional number of bits) does not affect the discernment of preceding and succeeding fields
PKD3-10 : include tests that verify that the fractional nature of the EOP does not affect the packet length determination.
TRD3,8,9 : include tests for all applicable packets ; include various lengths of data packets ranging up from 0.
TRD4-7 : include tests with timings outside the specified ranges on either side of the range. also include tests in which the timings are fractional number of bits.
TFD1-5 : include tests for all applicable transfers; include various lengths of data stage ranging up from 0.
BST1-4 : include various cases of placement of bitstream relative to the operative clock in the USB engine.
BST5 : should include thorough testing of the PLL or similar device used to interface between these two asynchronous domains.
BST6,7 : bitstuff error locations should include various locations in the packet/bitstream; include consecutive bitstuff errors.
FLT1-5 : include variety of errors and locations of errors.
FLT6,7 : the runt NIB0 should be located at various points in the packet/bitstream including the normal location of the EOP.
PKT1 : addresses used should include value 0 after address has been set to non-default value.
PKT6 : ISO endpoint data packets should either be prenegotiated size or 0 length only
PKT8 : EOP jitter can be seen as change in location of both edges of NIB 0 of the EOP; or as change in size of the NIB 0 and a change in its location in the packet. Either approach should include the effect of jitter induced by the frequency and phase differences between the two agents
TRT1 : for the purposes of this testing, a 'packet' definition can be broadened to include timeout and error in addition to unexpected packet (e.g. receiving ack when expecting data)

TFT1 : a bi-directional endpoint has a single data toggle value for both directions whose value is a function of the transfer stage and state of the data stage.
TFT2 : the error indication may be visible at the target only or can be communicated via stall handshake on bus.

Six perl programs are appended below. These programs are useful for converting between various notations (packets, nrz, nrzi) and for doing crc generation and checking and bit stuffing

```
pkt2nrzs:    converts a packet into a nrz bit stream
bsnrzs:      bit stuffs an nrz bit stream
bunrzis:     bit unstuffs an nrzi bit stream
tonrzi:      converts an nrz bit stream to an nrzi bitstream
tonrz:       converts an nrzi bit stream to an nrz bitstream
crc5:        generates token crc for nrz bit stream
crc16:       generates data crc for nrz bit stream
```

```
#! /usr/local/bin/perl
################################################################################
#
#     name:       pkt2nrzs
#     usage:      pkt2nrzs -i <input file> -o <output file>
#     desc:       this program will generate nrz bitstream for packets
in
#           input file.  Packet formats accepted are shown below. Fields
#           in the nrz bitstream are  separated by commas; order of
#           transmission is from left to right; lines are broken at 64
#           character boundaries. (note that the nrz stream is not bit
#            stuffed)
#           Blank lines or lines starting with # are ignored
#
#           PACKET            REQUIRED FIELDS
#           -------           ------
#           setup       addr <XX> endp <X> crc <XX>
#           out         addr <XX> endp <X> crc <XX>
#           in
#
#           sof         time <XXX> crc <XX>
#
#           data0       count <int> crc <XXXX>
#           data1       count <int> crc <XXXX>
#
#           ack
#           nack
#           stall
#           pre
#
#
#           legend:
#           X - hex digit
#           int - integer
#
#           The data packet count field specifies the number of bytes
#           to transfer.  The byte data will be an incrementing number
#           from 1 to count.
#
#
#           The parameter "gencrc" can be specified instead of crc <XX>
#           or crc <XXXX> to tell the compiler to generate a valid crc
#

#     Example input file and output file

#           out             addr   00 endp 0 crc ff
#
#           setup           addr ff endp f gencrc
#           # foll is a default address
#           in addr 00 endp 0 gencrc
#           sof             time 7ff gencrc
#           data0           count 2 crc ffff
#           data1           count  3 gencrc
#           ack
#           nack
#           stall
#           pre
```

```
#           00000001,10000111,0000000,0000,11111,XX1;
#           00000001,10110100,1111111,1111,00010,XX1;
#           00000001,10010110,0000000,0000,01000,XX1;
#           00000001,10100101,11111111111,00010,XX1;
#           00000001,11000011,10000000,01000000,1111111111111111, XX1;
#
      00000001,11010010,10000000,01000000,11000000,0111100101111001,XX
#           1;
#           00000001,01001011,XX1;
#           00000001,01011010,XX1;
#           00000001,01111000,XX1;
#           00000001,00111100;

################################################################################
########

# parse command line for flags

# PID constants
@out = ('1','0','0','0','0','1','1','1');
@in = ('1','0','0','1','0','1','1','0');
@sof = ('1','0','1','0','0','1','0','1');
@setup = ('1','0','1','1','0','1','0','0');
@data0 = ('1','1','0','0','0','0','1','1');
@data1 = ('1','1','0','1','0','0','1','0');
@ack = ('0','1','0','0','1','0','1','1');
@nak = ('0','1','0','1','1','0','1','0');
@stall = ('0','1','1','1','1','0','0','0');
@pre = ('0','0','1','1','1','1','0','0');

while ($arg = shift(@ARGV)) {
   if ($arg eq "-i") {
      $infile = shift(@ARGV);
   }
   if ($arg eq "-o") {
      $outfile = shift(@ARGV);
   }
}
if (defined($infile)) {
   open(INFILE,$infile) || die "ERROR: can't open $infile";
}
if (defined($outfile)) {
   open(OUTFILE,">$outfile") || die "ERROR: can't open $outfile";
}


sub hex2bin {
   local($num) = @_;
   local($count) = 0;
   local(@bits) = ();
   @digits = split(//,$num);
   while (@digits != ()) {
      $hexchar = shift(@digits);
      if ($hexchar =~ /0/) {
       @bits[$count..$count+3] = ('0','0','0','0');
      }
      elsif ($hexchar =~ /1/) {
       @bits[$count..$count+3] = ('0','0','0','1');
      }
```

```
      elsif ($hexchar =~ /2/) {
       @bits[$count..$count+3] = ('0','0','1','0');
      }
      elsif ($hexchar =~ /3/) {
       @bits[$count..$count+3] = ('0','0','1','1');
      }
      elsif ($hexchar =~ /4/) {
       @bits[$count..$count+3] = ('0','1','0','0');
      }
      elsif ($hexchar =~ /5/) {
       @bits[$count..$count+3] = ('0','1','0','1');
      }
      elsif ($hexchar =~ /6/) {
       @bits[$count..$count+3] = ('0','1','1','0');
      }
      elsif ($hexchar =~ /7/) {
       @bits[$count..$count+3] = ('0','1','1','1');
      }
      elsif ($hexchar =~ /8/) {
       @bits[$count..$count+3] = ('1','0','0','0');
      }
      elsif ($hexchar =~ /9/) {
       @bits[$count..$count+3] = ('1','0','0','1');
      }
      elsif ($hexchar =~ /a/i) {
       @bits[$count..$count+3] = ('1','0','1','0');
      }
      elsif ($hexchar =~ /b/i) {
       @bits[$count..$count+3] = ('1','0','1','1');
      }
      elsif ($hexchar =~ /c/i) {
       @bits[$count..$count+3] = ('1','1','0','0');
      }
      elsif ($hexchar =~ /d/i) {
       @bits[$count..$count+3] = ('1','1','0','1');
      }
      elsif ($hexchar =~ /e/i) {
       @bits[$count..$count+3] = ('1','1','1','0');
      }
      elsif ($hexchar =~ /f/i) {
       @bits[$count..$count+3] = ('1','1','1','1');
      }
      $count = $count + 4;
   }
   return @bits;
}


sub xor5 {
   local(@x) = @_[0..4];
   local(@y) = @_[5..9];
   local(@results5) = ();
   for($j=0;$j<5;$j++)  {
      if (shift(@x) eq shift(@y)) { push(@results5, '0'); }
      else { push(@results5, '1'); }
                        }
   return(@results5[0..4]);
}

sub xor16 {
```

```
   local(@x) = @_[0..15];
   local(@y) = @_[16..31];
   local(@results16) = ();
   for($j=0;$j<16;$j++)         {
      if (shift(@x) eq shift(@y)) { push(@results16, '0'); }
      else { push(@results16, '1'); }
                                }
   return(@results16[0..15]);
}


sub crc5 {
   local($st_data) = @_;
   local(@G) =    ('0','0','1','0','1');
   local(@data) = split (//,$st_data);
   local(@hold) = ('1','1','1','1','1');
   if (scalar(@data) > 0)       {
   loop5: while (scalar(@data) > 0)     {
if (shift(@data) eq shift(@hold)) {push(@hold, '0')}
else    { push(@hold, '0'); @hold = &xor5(@hold,@G); }
                                        }
                                }
for ($i=0;$i<=$#hold;$i++) {if (@hold[$i] eq "1") {@hold[$i]=0} else
{@hold[$i]=1}}
   return(@hold);
}

sub crc16 {
   local($st_data) = @_;
   local(@G) =
('1','0','0','0','0','0','0','0','0','0','0','0','0','1','0','1');
   local(@data) = split (//,$st_data);
   local(@hold) =
('1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1');
   if (scalar(@data) > 0)       {
   loop16: while (scalar(@data) > 0)    {
if (shift(@data) eq shift(@hold)) {push(@hold, '0')}
else    { push(@hold, '0'); @hold = &xor16(@hold,@G); }
                                        }
                                }
for ($i=0;$i<=$#hold;$i++) {if (@hold[$i] eq "1") {@hold[$i]=0} else
{@hold[$i]=1}}
   return(@hold);
}


$line_count = 0;

NL:while (<INFILE>) {
#print $_;
    tr/A-Z/a-z/;  # make everything lowercase
    $line_count++;
    $line = $_;
    if ($line eq "\n" ) {next NL} ## blank line
    if ($line =~ /^\#/) {next NL} ## comment
    $count = 9;
    # sync pattern begins every packet
    @bit_stream[0..8] = ('0','0','0','0','0','0','0','1',',');
    chop;
    @tokens = split;
```

```
$current = shift(@tokens);
if ($current =~ /setup/i) {
   @bit_stream[$count..$count+7] = @setup;
   $packet = "token";
}
if ($current =~ /out/i) {
   @bit_stream[$count..$count+7] = @out;
   $packet = "token";
}
elsif ($current =~ /in/i) {
   @bit_stream[$count..$count+7] = @in;
   $packet = "token";
}
elsif ($current =~ /pre/i) {
   @bit_stream[$count..$count+7] = @pre;
   $packet = "pre";
}
elsif ($current =~ /sof/i) {
   @bit_stream[$count..$count+7] = @sof;
   $packet = "sof";
}
elsif ($current =~ /data0/i) {
   @bit_stream[$count..$count+7] = @data0;
   $packet = "data";
}
elsif ($current =~ /data1/i) {
   @bit_stream[$count..$count+7] = @data1;
   $packet = "data";
}
elsif ($current =~ /\back\b/i) {
   @bit_stream[$count..$count+7] = @ack;
   $packet = "hand";
}
elsif ($current =~ /nack/i) {
   @bit_stream[$count..$count+7] = @nak;
   $packet = "hand";
}
elsif ($current =~ /stall/i) {
   @bit_stream[$count..$count+7] = @stall;
   $packet = "hand";
}

   $count = $count + 8;
   if ($packet ne "pre") {$bit_stream[$count] = ','; $count++}

if ($packet eq "token") {
   $current = shift(@tokens);
   if ($current =~ /addr/i) {
      $current = shift(@tokens);
    @bit_string = &hex2bin($current);
    @addr_st = reverse(@bit_string[1..7]);
      @bit_stream[$count..$count+6] = reverse(@bit_string[1..7]);
      $count = $count + 7;
      $bit_stream[$count] = ',';
      $count++;
   }
   else {
    die "ERROR: unexpected input <$current> on line $line_count\n";
   }
   $current = shift(@tokens);
```

```
        if ($current =~ /endp/i) {
           $current = shift(@tokens);
         @bit_string = &hex2bin($current);
         @addr_st = join('',@addr_st,reverse(@bit_string[0..3])) ;
           @bit_stream[$count..$count+3] = reverse(@bit_string[0..3]);
           $count = $count + 4;
           $bit_stream[$count] = ',';
           $count++;
        }
        else {
         die "ERROR: unexpected input <$current> on line $line_count\n";
        }
        $current = shift(@tokens);
        if ($current =~ /\bcrc\b/i) {
           $current = shift(@tokens);
         @bit_string = &hex2bin($current);
           @bit_stream[$count..$count+4] = reverse(@bit_string[3..7]);
           $count = $count + 5;
           $bit_stream[$count] = ',';
           $count++;
        }
        elsif ($current =~ /gencrc/i) {
           @bit_string = &crc5(@addr_st);
           @bit_stream[$count..$count+4] = @bit_string;
           $count = $count + 5;
           $bit_stream[$count] = ',';
           $count++;
        }
        else {
         die "ERROR: unexpected input <$current> on line $line_count\n";
        }
        $bit_stream[$count] = ‘x’;
        $count++;
        $bit_stream[$count] = ‘x’;
        $count++;
        $bit_stream[$count] = '1';
        $count++;
    }
    elsif ($packet eq "sof") {
        $current = shift(@tokens);
        if ($current =~ /time/i) {
           $current = shift(@tokens);
         @bit_string = &hex2bin($current);
           @sof_st = reverse(@bit_string[1..10]);
           @sof_st = join('',@bit_string[11],@sof_st); ## do this to
convert array sof_st into string
           @bit_stream[$count..$count+10] = reverse(@bit_string[1..11]);
           $count = $count + 11;
           $bit_stream[$count] = ',';
           $count++;
        }
        else {
         die "ERROR: unexpected input <$current> on line $line_count\n";
        }
        $current = shift(@tokens);
        if ($current =~ /\bcrc\b/i) {
           $current = shift(@tokens);
         @bit_string = &hex2bin($current);
           @bit_stream[$count..$count+4] = reverse(@bit_string[3..7]);
           $count = $count + 5;
```

```
            $bit_stream[$count] = ',';
            $count++;
         }
         elsif ($current =~ /gencrc/i) {
            @bit_string = &crc5(@sof_st);
            @bit_stream[$count..$count+4] = @bit_string;
            $count = $count + 5;
            $bit_stream[$count] = ',';
            $count++;
         }
         else {
          die "ERROR: unexpected input <$current> on line $line_count\n";
         }
         $bit_stream[$count] = ‘x’;
         $count++;
         $bit_stream[$count] = ‘x’;
         $count++;
         $bit_stream[$count] = '1';
         $count++;
      }
      elsif ($packet eq "data") {
         $current = shift(@tokens);
         if ($current =~ /count/i) {
            $num_bytes = shift(@tokens);
          $data_count = 1;
            @data_string = ();
          while ($data_count <= $num_bytes) {
             $hex_byte = sprintf("%2x", $data_count);
             $hex_byte =~ s/ /0/;
             @bit_string = &hex2bin($hex_byte);
               @data_string =
join('',@data_string,reverse(@bit_string[0..7]));
               @bit_stream[$count..$count+7] = reverse(@bit_string[0..7]);
             $data_count++;
               $count = $count + 8;
               $bit_stream[$count] = ',';
               $count++;
          }
         }
         else {
          die "ERROR: unexpected input <$current> on line $line_count\n";
         }
         $current = shift(@tokens);
         if ($current =~ /\bcrc\b/i) {
            $current = shift(@tokens);
          @bit_string = &hex2bin($current);
            @bit_stream[$count..$count+15] = reverse(@bit_string[0..15]);
            $count = $count + 16;
            $bit_stream[$count] = ',';
            $count++;
         }
         elsif ($current =~ /gencrc/i) {
            @bit_string = &crc16(@data_string);
            @bit_stream[$count..$count+15] = @bit_string[0..15];
            $count = $count + 16;
            $bit_stream[$count] = ',';
            $count++;
         }
         else {
          die "ERROR: unexpected input <$current> on line $line_count\n";
```

```
        }
        $bit_stream[$count] = ‘x’;
        $count++;
        $bit_stream[$count] = ‘x’;
        $count++;
        $bit_stream[$count] = '1';
        $count++;
     }
     elsif ($packet eq "hand") {
        $bit_stream[$count] = ‘x’;
        $count++;
        $bit_stream[$count] = ‘x’;
        $count++;
        $bit_stream[$count] = '1';
        $count++;
     }

     $bit_stream[$count] = ';';

    #print bit stream to output file
    for ($i=0; $i <= $count; $i++) {
       if (($i % 64) == 0 && $i > 0) {
          print OUTFILE "\n";
          print OUTFILE $bit_stream[$i];
        if ($bit_stream[$i] eq ';') {
           print OUTFILE "\n";
        }
       }
       else {
          print OUTFILE $bit_stream[$i];
        if ($bit_stream[$i] eq ';') {
           print OUTFILE "\n";
        }
       }
    }
}
 #! /usr/local/bin/perl
## bit stuffs nrz stream
## usage: bsnrzs
## NZBstream
## bit stream is read from left to right

$count=0;

while (<>){
tr/A-Z/a-z/;        # make everything lowercase
print $_; ## print NZB stream
@strm=split(//);
for($i=0;$i<$#strm;$i++){
print $strm[$i];
if ($strm[$i]eq"1") { $count++; if ($count == 6) { print "0"; $count=0}
next}
if ($strm[$i]eq"0")  {$count=0;next}
if ($strm[$i]eq"x") { $count=0;next}
next;  ## if NZB stream has some non0/1/x character e.g. space
}
print "\n";
      }
```

```
#! /usr/local/bin/perl
## bit unstuffs nrzi stream
## usage: bunrzis
## NIBstream
## bit stream is read from left to right

$count=0;

while (<>){
tr/A-Z/a-z/;        # make everything lowercase
print $_; ## print NZB stream
@strm=split(//);
for($i=0;$i<$#strm;$i++){
if ($strm[$i]eq"0")  {if ($count == 6) {$count=0} else { $count=0;print
"0"};next}
if ($strm[$i]eq"1") { if ($count==6) {print "\nbitstuff error\n"; $count
= 0} else {print "1";$count++;}  next}
if ($strm[$i]eq"x") { if ($count==6) {print "\nbitstuff error\n"; $count
= 0} else {print "X";$count=0;}  next}
print $strm[$i];next;  ## if NZB stream has some non0/1/x character e.g.
space
}
print "\n";
      }



#! /usr/local/bin/perl
## converts nrz stream to nrzi stream
## usage: tonrzi
## NZBstream
## assumes bus in idle J at start

$prev="J";

while (<>){
tr/A-Z/a-z/;        # make everything lowercase
print $_; ## print NZB stream
@strm=split(//);
for($i=0;$i<$#strm;$i++){
if ($strm[$i]eq"1") { print $prev; next}  ## no change of state
if ($strm[$i]eq"0")  {if ($prev eq "J") {print "K";$prev="K";next;} if
($prev eq "K") {print "J";$prev="J";next;} }
if ($strm[$i]eq"x") { print $strm[$i]; $prev = "J"; next} ## se0=x means
next is J
print $strm[$i]; next;  ## if NZB stream has some non0/1/x character
e.g. space
}
print "\n";
      }



#! /usr/local/bin/perl
## converts nrzi stream to nrz stream
## usage: tonrz
## NIBstream
```

```
## assumes bus in idle j at start

$prev="j";

while (<>){
tr/A-Z/a-z/;        # make everything lowercase
print $_; ## print NIB stream
@strm=split(//);
for($i=0;$i<$#strm;$i++){
if (($strm[$i] eq "j") || ($strm[$i] eq "k"))
      { if ($strm[$i] eq $prev ) {print "1"} else {print "0"}
      $prev= $strm[$i]; next
      }
if ($strm[$i]eq"0")  {print "X"; $prev =  "j"; next}  ## because EOP is
XX1 or 00j
print $strm[$i]; next;  ## if NIB stream has some nonj/k/0 character
e.g. space
}
print "\n";
      }



#! /usr/local/bin/perl
## crc5 nrzstream
## nrz stream is sent in left to right order
## generated crc should also be sent out in left to right order

sub xor5 {
   local(@x) = @_[0..4];
   local(@y) = @_[5..9];
   local(@results5) = ();
   for($j=0;$j<5;$j++)  {
      if (shift(@x) eq shift(@y)) { push(@results5, '0'); }
      else { push(@results5, '1'); }
                  }
   return(@results5[0..4]);
}

          {
   local($st_data) = $ARGV[0];
   local(@G) =    ('0','0','1','0','1');
   local(@data) = split (//,$st_data);
   local(@hold) = ('1','1','1','1','1');
   if (scalar(@data) > 0)     {
   loop5: while (scalar(@data) > 0)       {
$nextb=shift(@data);
if (($nextb ne "0") && ($nextb ne "1")) {next loop5} ## comment
character
if ($nextb eq shift(@hold)) {push(@hold, '0')}
else  { push(@hold, '0'); @hold = &xor5(@hold,@G); }
##   print (@hold); print "\n";
                              }
                        }
##   print (@hold); print "\n";
## invert shift reg contents to generate crc field
for ($i=0;$i<=$#hold;$i++) {if (@hold[$i] eq "1") {print("0")} else {
print("1")} }
print "\n";
```

```
            }


#! /usr/local/bin/perl
## usage:
## crc16 nrzstream
## nrz stream is sent in left to right order
## generated crc should also be sent out in left to right order

sub xor16 {
   local(@x) = @_[0..15];
   local(@y) = @_[16..31];
   local(@results16) = ();
   for($j=0;$j<16;$j++)       {
      if (shift(@x) eq shift(@y)) { push(@results16, '0'); }
      else { push(@results16, '1'); }
                           }
   return(@results16[0..15]);
}
            {
   local($st_data) = $ARGV[0];
   local(@G) =
('1','0','0','0','0','0','0','0','0','0','0','0','0','1','0','1');
   local(@hold) =
('1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1');
   local(@data) = split (//,$st_data);
   if (scalar(@data) > 0)     {
   loop16: while (scalar(@data) > 0)      {
$nextb=shift(@data);
if (($nextb ne "0") && ($nextb ne "1")) {next loop16} ## comment
character
if ($nextb eq shift(@hold)) {push(@hold, '0')}
else  { push(@hold, '0'); @hold = &xor16(@hold,@G); }
                              }
                           }
#   print (@hold); print "\n";
## invert shift reg to generate CRC field
for ($i=0;$i<=$#hold;$i++) {if (@hold[$i] eq "1") {print("0")} else {
print("1")} }
print "\n";
            }
```